CS1001.py

Extended Introduction to Computer Science with Python, Tel-Aviv University, Spring 2013

Recitation 3 - 14-18.3.2013

Last update: 18.3.2013

Divisors

In the previous recitation we wrote a function to find the divisors of a number:

```
def divisors(n):
    return [div for div in range(1,n) if n % div == 0]
```

Here is a faster and slightly more complex way to do it:

```
from math import ceil
def divisors2(n):
    divs = [1]
    for m in range(2, ceil(n ** 0.5)):#1 and n**0.5 will be handled separately. why?
        if n % m == 0:
            divs += [m, n // m]
    if n % n ** 0.5 == 0:
        divs += [int(n ** 0.5)]
    return divs

print(divisors(36))
print(sorted(divisors2(36)))
```

```
[1, 2, 3, 4, 6, 9, 12, 18]
[1, 2, 3, 4, 6, 9, 12, 18]
```

Timing operations

Here is the simplest way to measure the time an operation takes. This method uses the `clock` function of the `time` module. It is the simplest way to do it and as such it is a crude way of doing it with very little statistical power and significance.

```
import time
help(time.clock)

Help on built-in function clock in module time:

clock(...)
    clock() -> floating point number

    Return the CPU time or real time since the start of the process or since
    the first call to clock().  This has as much precision as the system
    records.

print(time.clock())
print(time.clock())

25.99883061412379
25.99907632401105
```

This way of timing operations is often called the *tic-toc* way, we save the time before and after the operation and subtract to find the time interval. Run this a few times to see how crude it is.

```
n = 1234567890
tic = time.clock()
divisors(n)
toc = time.clock()
print("divisors: ",(toc-tic))
tic = time.clock()
divisors2(n)
toc = time.clock()
print("divisors2:",(toc-tic))

divisors:  467.4390757203012
divisors2: 0.014307922181728827
```

The binary system and base conversions

A binary number is a number in the base 2, which means that it only uses 2 digits - 0 and 1. The "regular" numbers we use, the decimal numbers, are in base 10, which means they use 10 digits - 0,1,2,3,4,5,6,7,8,9.

What is a base? To understand base X imagine you have X fingers instead of 10. **How would you count with X fingers?**

Converting binary to decimal

Looking at a binary number, 10011010, the **Least Significant Digit** (or **bit** for binary digits), in this case 0, is the right most digit, and if it is 1 then it is worth $2^0 = 1$, otherwise it is worth 0. The next bit (in this case 1) is worth $2^1 = 2$. The next one is worth $2^2 = 4$, and the k-th digit/bit from the right (starting with k=*0*) is worth 2^k. In general, denoting the binary number $x_{base2} = a_n a_{n-}...a_1 a_0$, it's decimal value can be evaluated by

$$x_{base10} = \sum_{n \geq k \geq 0} a_k 2^k$$

. Let's write python code for this:

```
x_bin = "11110"
x_bin = x_bin[::-1] # reverse it so that LSB is on the left for the iteration
x_dec = 0
for k in range(len(x_bin)):
    bit = int(x_bin[k])
    print(k,bit)
    x_dec += bit * 2**k
print(x_dec)
```

```
0 0
1 1
2 1
3 1
4 1
30
```

Converting decimal to binary

Converting from decimal to binary is done by integer division. Remember that taking the modulo 10 of a number gives the LSD in base 10, and diving by 10 removes the LSD. This is the basic idea:

```
x_dec = 42
x_bin = ''
while x_dec > 0:
    bit = x_dec % 2
    x_bin = str(bit) + x_bin
    x_dec = x_dec // 2
print(x_bin)
```

```
101010
```

Builtin functions

There are some python functions to do these operations:

```
bin(42)
```

```
'0b101010'
```

```
int('101010',2)
```

```
42
```

You can also use base 16 - hexadecimal numbers:

```
hex(42)
```

```
'0x2a'
```

```
int('2a',16)
```

```
42
```

General conversion

We want to convert from base 10 to base b ($2 \leq b < 10$) :

```
def convert_base(n,b):
    '''convert_base(integer, integer) -> string
    Return the textual representation of n (decimal) in base 2 <= b <= 10.
    '''
    result = ''
    while n > 0:
        digit = n % b
        n = n // b
        print(digit)
        result = str(digit) + result
    return result
```

```
convert_base(23,12)
```

```
1+12+12**2
```

```
convert_base(10,16)
```

and now to base b for $10 < b \leq 36$:

```
def convert_base(n,b):
    '''convert_base(integer, integer) -> string
    Return the textual representation of n (decimal) in base 2 <= b <= 10.
    '''
    assert 2 <= b <= 36

    if n == 0:
        result = '0'
    elif n < 0:
        result = '-'
    else:
        result = ''
    n = abs(n)
    while n > 0:
        digit = n % b
        n = n // b
        # str(digit) only works for b <= 10
        result = '0123456789abcdefghijklmnopqrstuvwxyz'[digit] + result
    return result

convert_base(23,12)

convert_base(10,6)

convert_base(10,16)

convert_base(40,32)

convert_base(0,5)

convert_base(100,55)
```

Python's memory model

See more examples at the Python Tutor website.

Fin

This notebook is part of the Extended introduction to computer science course at Tel-Aviv University.

The notebook was written using Python 3.2 and IPython 0.13.1.

The code is available at https://raw.github.com/yoavram/CS1001.py/master/recitation3.ipynb.

The notebook can be viewed online at http://nbviewer.ipython.org/urls/raw.github.com/yoavram/CS1001.py/master/recitation3.ipynb.

The notebooks is also available as a PDF at https://github.com/yoavram/CS1001.py/blob/master/recitation3.pdf?raw=true.


```
!python d:\\workspace\\nbconvert\\nbconvert.py -f latex d:\\university\\CS1001.py\\recitatio
!pandoc d:\\university\\CS1001.py\\recitation3.tex -o d:\\university\\CS1001.py\\recitation3
```